PARTE

I

EL PRODUCTO Y EL PROCESO

EN esta parte de *Ingeniería del software: un enfoque práctico* aprenderá sobre el producto que va a ser tratado con ingeniería y el proceso que proporciona un marco de trabajo para la tecnología de Ingeniería del software. En los capítulos siguientes se tratan las preguntas:

- ¿Qué es realmente el software de computadora?
- ¿Por qué se lucha para construir sistemas de alta calidad basados en computadoras?
- ¿Cómo se pueden establecer categorías de dominios de aplicaciones para software de computadoras?
- ¿Qué mitos de software van a existir?
- ¿Qué es un «proceso» de software?
- ¿Existe una forma genérica de evaluar la calidad de un proceso?
- ¿Qué modelos de procesos se pueden aplicar al desarrollo del software?
- ¿En que difieren los modelos de proceso lineales e iterativos?
- ¿Cuáles son sus puntos fuertes y débiles?
- ¿Qué modelos de proceso avanzados se han propuesto para la ingeniería del software?

Una vez contestadas todas estas preguntas, estará más preparado para comprender los aspectos técnicos y de gestión de la disciplina de ingeniería a la que se dedica el resto del libro.

CAPÍTULO

1 EL PRODUCTO

LAS alarmas comenzaron más de una década antes del acontecimiento. Con menos de dos años a la fecha señalada, los medios de comunicación recogieron la historia. Los oficiales del gobierno expresaron su preocupación, los directores de la industria y de los negocios comprometieron grandes cantidades de dinero, y por Último, las advertencias horribles de catástrofe llegaron a la conciencia del público. El software, al igual que el ahora famoso error Y2K, podría fallar, y como resultado, detener el mundo como nosotros lo conocimos.

Como vimos durante los últimos meses del año 1999, sin querer, no puedo dejar de pensar en el párrafo profético contenido en la primera página de la cuarta edición de este libro. Decía:

> El software de computadora se ha convertido en el ***alma*** *mater*. Es la máquina que conduce a la toma de decisiones comerciales. Sirve de base para la investigación científica moderna y de resolución de problemas de ingeniería. Es el factor clave que diferencia los productos y servicios modernos. Está inmerso en sistemas de todo tipo: de transportes, médicos, de telecomunicaciones, militares, procesos industriales, entretenimientos, productos de oficina..., la lista es casi interminable. El software **es** casi ineludible en un mundo moderno. **A** medida que nos adentremos en el siglo XXI, será el que nos conduzca a nuevos avances en todo, desde la educación elemental a la ingeniería genética.

VISTAZO RÁPIDO

¿Qué es? El software de computadora es el producto que diseñan y construyen los ingenieros del software. Esto abarca programas que se ejecutan dentro de una computadora de cualquier tamaño y arquitectura, documentos que comprenden formularios virtuales e impresos y datos que combinan números y texto y también incluyen representaciones de información de audio, vídeo e imágenes.

¿Quién lo hace? Los ingenieros de software lo construyen, y virtualmente cualquier persona en el mundo industrializado lo utiliza bien directa o indirectamente.

¿Por qué es importante? Porque afecta muy de cerca a cualquier aspecto de nuestra vida y está muy extendido en nuestro comercio, cultura y en nuestras actividades cotidianas.

¿Cuáles son los pasos? Construir software de computadora como construimos cualquier otro producto satisfactorio, aplicando un proceso que conduce a un resultado de alta calidad que satisface las necesidades de la gente que usará el producto. Debes aplicar un enfoque de ingeniería de software.

¿Cuál es el producto obtenido? Desde el punto de vista de un ingeniero de software, el producto obtenido son los programas, documentos y los datos que configuran el software de computadora. Pero desde el punto de vista de los usuarios el producto obtenido es la información resultante que hace de algún modo el mundo mejor a los usuarios.

¿Cómo puedo estar seguro de que lo he hecho correctamente? Lee el resto de este libro, selecciona aquellas ideas que son aplicables al software que construyes y aplícalas a tu trabajo.

Cinco años después de que la cuarta edición de este libro fue escrita, el papel del software como «alma mater» ha llegado a ser más obvio. Un director de software de Intemet ha producido su propia economía de 500 billones de Euros. En la euforia creada por la promesa de un paradigma económico nuevo, los inversores de Wall Street dieron a las pequeñas empresas «punto-com» estimaciones en billones de dólares antes de que éstas comenzasen a producir un dólar en ventas. Han surgido nuevas industrias dirigidas por software y las antiguas que no se han adaptado a esta nueva tendencia están ahora amenazadas de extinción. El gobierno de Estados Unidos ha mantenido un contencioso frente a la mayor compañía de la industria del software, como lo mantuvo hace poco tiempo cuando se movilizó para detener las actividades monopolísticas en las industrias del acero y del aceite.

El impacto del software en nuestra sociedad y en la cultura continúa siendo profundo. Al mismo tiempo que crece su importancia, la comunidad del software trata continuamente de desarrollar tecnologías que hagan más sencillo, rápido y menos costosa la construcción de programas de computadora de alta calidad.

Este libro presenta un marco de trabajo que puede ser usado por aquellos que construyen software informático —aquellos que lo deben hacer bien—. La tecnología que comprende un proceso, un juego de métodos y un conjunto de herramientas **se** llama *ingeniería **del** software.*

1.1. LA EVOLUCIÓN DEL SOFTWARE

Hoy en día el software tiene un doble papel. Es un producto y, al mismo tiempo, el vehículo para entregarlo. Como producto, hace entrega de la potencia informática que incorpora el hardware informático o, más ampliamente, una red de computadoras que es accesible por hardware local. Si reside dentro de un teléfono celular u opera dentro de una computadora central, el software es un transformador de información, produciendo, gestionando, adquiriendo, modificando, mostrando o transmitiendo información que puede ser tan simple como un solo bit, o tan complejo como una presentación en multimedia. Como vehículo utilizado para hacer entrega del producto, el software actúa como la base de control de la computadora (sistemas operativos), la comunicación de información (redes) y la creación y control de otros programas (herramientas de software y entomos).

PUNTO CLAVE

El software es tonto un producto, como el vehículo poro su entrego

El papel del software informático ha sufrido un cambio significativo durante un periodo de tiempo superior a 50 años. Enormes mejoras en rendimiento del hardware, profundos cambios de arquitecturas informáticas, grandes aumentos de memoria y capacidad de almacenamiento y una gran variedad de opciones de entrada y salida han conducido a sistemas más sofisticados y más complejos basados en computadora. La sofisticación y la complejidad pueden producir resultados deslumbrantes cuando un sistema tiene éxito, pero también pueden suponer grandes problemas para aquellos que deben construir sistemas complejos.

Libros populares publicados durante los años 70 **y** 80 proporcionan una visión histórica útil dentro de la percepción cambiante de las computadoras y del software, **y** de **su** impacto en nuestra cultura. Osborne [OSB79] hablaba de una «nueva revolución industrial». Toffler [TOF80] llamó a la llegada de componentes microelectrónicos la «tercera ola del cambio» en la historia de la humanidad, **y** Naisbitt [NAI82] predijo la transformación de la sociedad industrial a una «sociedad de información». Feigenbaum y McCorduck [FEI83] sugirieron que la información **y** el conocimiento (controlados por computadora) serían el foco de poder del siglo veintiuno, y Stoll [STO89] argumentó que la «comunidad electrónica» creada mediante redes **y** software es la clave para el intercambio de conocimiento alrededor del mundo.

Al comienzo de los años 90, Toffler [TOF90] describió un «cambio de poder» en el que las viejas estructuras de poder (gubernamentales, educativas, industriales, económicas **y** militares) se desintegrarían a medida que

Cita:

Me introduje en el futuro, más allá de lo que el ojo humano puede ver. Tuve una visión del mundo y todo lo maravilloso que podría ser.
Tennyson

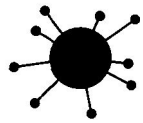
las computadoras y el software nos llevaran a la «democratización del conocimiento». A Yourdon [YOU92] le preocupaba que las compañías en Estados Unidos pudieran perder **su** competitividad en empresas relativas al software y predijo «el declive y la caída del programador americano». Hammer y Champy [HAM93] argumentaron que las tecnologías de información iban a desempeñar el papel principal en la «reingeniería de la compañía». A mediados de los años 90, la persistencia de las computadoras y del software generó una erupción de libros por «neo-Luddites» (por ejemplo: *Resisting the Virtual Life,* editado por James Brook y Ian Boal, y *The Future Does not Compute* de Stephen Talbot). Estos autores critican enormemente la computadora, haciendo énfasis en preocupaciones legítimas pero ignorando los profundos beneficios que se han llevado a cabo [LEV95].

Cita:

Las computadoras hacen las cosas más fáciles, pero la mayoría de las cosas que facilitan no es preciso hacerlas.
Andy Rooney

Al final de los años 90, Yourdon [YOU96] volvió a evaluar las perspectivas del software profesional y sugirió la «resurrección y elevación» del programador americano. A medida que internet creció en importancia, **su** cambio de pensamiento demostró ser correcto. Al final del siglo veinte, el enfoque cambió una vez más. Aquí tuvo lugar el impacto de la «bomba de relojería» Y2K (por ejemplo: [YOU98b], [DEJ98], [KAR99]). Aunque muchos vieron las predicciones de los críticos del Y2K como reacciones, sus populares lecturas devolvieron la difusión del software a sus vidas. Hoy en día, «la computación omnipresente» [NOR98] ha producido una generación de aplicaciones de información que tienen conexión en banda ancha a la Web para proporcionar «una capa de conexión sobre nuestras casas, oficinas, y autopistas» [LEV99]. El papel del software continúa su expansión.

El programador solitario de antaño ha sido reemplazado por un equipo de especialistas del software, cada uno centrado en una parte de la tecnología requerida para entregar una aplicación concreta. Y de este modo, las cuestiones que se preguntaba el programador solitario son las mismas cuestiones que nos preguntamos cuando construimos sistemas modernos basados en computadoras:



Estadísticas globales de software

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué son tan elevados los costes de desarrollo?
- ¿Por qué no podemos encontrar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué nos resulta difícil constatar el progreso conforme se desarrolla el software?

1.2 EL SOFTWARE

En 1970, menos del uno por ciento de las personas podría haber descrito inteligentemente lo que significaba «software de computadora». Hoy, la mayoría de los profesionales y muchas personas en general piensan en su mayoría que comprenden el software. ¿Pero lo entienden realmente?

1.2.1. Características del software

Para poder comprender lo que es el software (y consecuentemente la ingeniería del software), es importante examinar las características del software que lo diferencian de otras cosas que los hombres pueden construir. Cuando se construye hardware, el proceso creativo humano (análisis, diseño, construcción, prueba) se traduce finalmente en una forma física. Si construimos una nueva computadora, nuestro boceto inicial, diagramas formales de diseño y prototipo de prueba, evolucionan hacia un producto físico (chips, tarjetas de circuitos impresos, fuentes de potencia, etc.).

El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto el software tiene unas características considerablemente distintas a las del hardware:

PUNTO CLAVE

El software se desarrolla, no se fabrica.

1. El software se desarrolla, no se fabrica en un sentido clásico.

Aunque existen similitudes entre el desarrollo del software y la construcción del hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen (o son fácilmente corregibles) en el software. Ambas actividades dependen de las personas, pero la relación entre las personas dedicadas y el trabajo realizado es completamente diferente para el software (véase el Capítulo 7). Ambas actividades requieren la construcción de un «producto» pero los enfoques son diferentes.

Los costes del software se encuentran en la ingeniería. Esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación.

2. ***El software no se «estropea».***

La Figura **1.1** describe, para el hardware, la proporción de fallos como una función del tiempo. Esa relación, denominada frecuentemente «curva de bañera», indica que el hardware exhibe relativamente muchos fallos al principio de su vida (estos fallos son atribuibles normalmente a defectos del diseño o de la fabricación); una vez corregidos los defectos, la tasa de fallos cae hasta un nivel estacionario (bastante bajo, con un poco de optimismo) donde permanece durante un cierto periodo de tiempo. Sin embargo, conforme pasa el tiempo, el hardware empieza a desgastarse y la tasa de fallos se incrementa.

El software no es susceptible a los males del entorno que hacen que el hardware se estropee. Por tanto, en teoría, la curva de fallos para el software tendría la forma que muestra la Figura **1.2.** Los defectos no detectados haran que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen (suponiendo que no se introducen nuevos errores) la curva se aplana, como se muestra. La curva idealizada es una gran simplificación de los modelos reales de fallos del software (vease más información en el Capítulo **8).** Sin embargo la implicación es clara, el software no se estropea. ¡Pero se deteriora!

PUNTO CLAVE

El software no se estropea, pero se deteriora.

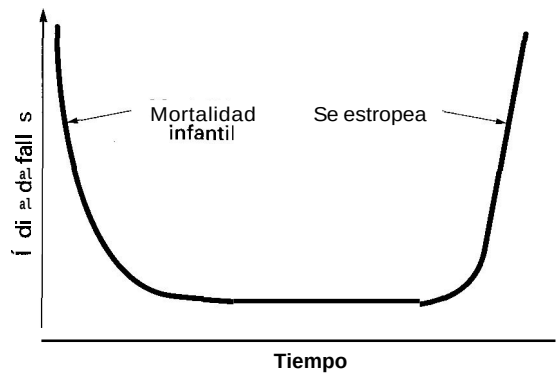


FIGURA 1.1. Curva de fallos del hardware.

Esto que parece una contradicción, puede comprenderse mejor considerando «la curva actual» mostrada en la Figura 1.2. Durante su vida, el software sufre cambios (mantenimiento). Conforme se hacen los cambios, es bastante probable que se introduzcan nuevos defectos, haciendo que la curva de fallos tenga picos como se ve en la Figura 1.2. Antes de que la curva pueda volver al estado estacionario original, se solicita otro cambio, haciendo que de nuevo se cree otro pico. Lentamente, el nivel mínimo de fallos comienza a crecer —el software se va deteriorando debido a los cambios—.

Otro aspecto de ese deterioro ilustra la diferencia entre el hardware y el software. Cuando un componente de hardware se estropea se sustituye por una pieza de repuesto. No hay piezas de repuesto para el software. Cada fallo en el software indica un error en el diseño o en el proceso mediante el que se tradujo el diseño a código máquina ejecutable. Por tanto, el mantenimiento del software tiene una complejidad considerablemente mayor que la del mantenimiento del hardware.

3. *Aunque la industria tiende a ensamblar componentes, la mayoría del software se construye a medida.*

Consideremos la forma en la que se diseña y se construye el hardware de control para un producto basado en computadora. El ingeniero de diseño construye un sencillo esquema de la circuitería digital, hace algún análisis fundamental para asegurar que se consigue la función adecuada y va al armario donde se encuentran los catálogos de componentes digitales. Después de seleccionar cada componente, puede solicitarse la compra.

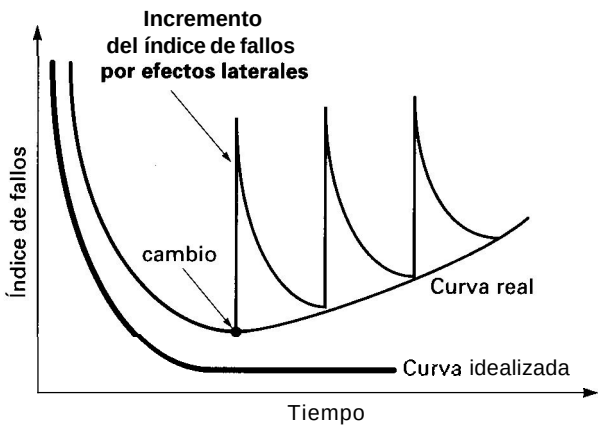


FIGURA 1.2. Curvas de fallos real e idealizada del software.

PUNTO CLAVE

Los métodos de ingeniería de software se esfuerzan para reducir la magnitud de los picos y la inclinación de la curva (Fig. 1.2).

A medida que la disciplina del software evoluciona, se crea un grupo de componentes de diseño estándar. Tornillos estándar y circuitos integrados preparados para la venta son solamente los dos mil coinponentes estándar que utilizan ingenieros mecánicos y eléctricos cuando diseñan nuevos sistemas. Los componentes reutilizables se han creado para que el ingeniero pueda concentrarse en elementos verdaderamente innovadores de un diseño, por ejemplo, las partes del diseño que representan algo nuevo. En el mundo del hardware, la reutilización de componentes es una parte natural del proceso de ingeniería. En el mundo del software es algo que sólo ha comenzado a lograrse en una escala amplia.

El componente de software debería diseñarse e implementarse para que pueda volver a ser reutilizado en muchos programas diferentes. En los años 60, se construyeron bibliotecas de subrutinas científicas reutilizables en una amplia serie de aplicaciones científicas y de ingeniería. Esas bibliotecas de subrutinas reutilizaban de forma efectiva algoritmos bien definidos, pero tenían un dominio de aplicación limitado. Hoy en día, hemos extendido nuestra visión de reutilización para abarcar no sólo los algorítmos, sino también estructuras de datos. Los componentes reutilizables modernos encapsulan tanto datos como procesos que se aplican a los datos, permitiendo al ingeniero del software crear nuevas aplicaciones a partir de las partes reutilizables. Por ejemplo, las interfaces gráficas de usuario de hoy en día se construyen frecuentemente a partir de componentes reutilizables que permiten la creación de ventanas gráficas, de menús despleglables y de una amplia variedad de mecanismos de interacción.

C VE

La mayoría del software sigue construyéndose a medida.

1.2.2. Aplicaciones del software

El software puede aplicarse en cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales (es decir, un algoritmo) (excepciones notables a esta regla son el software de los sistemas expertos y de redes neuronales). El contenido y el determinismo de la información son factores importantes a considerar para determinar la naturaleza de una aplicación de software. El contenido se refiere al significado y a la forma de la información de entrada y salida. Por ejemplo, muchas aplicaciones bancarias usan unos datos de entrada muy estructurados (una base de datos) y producen «informes» con determinados formatos. El software que controla una máquina automática (por ejemplo: un control numérico) acepta elementos de datos discretos con una estructura limitada y produce órdenes concretas para la máquina en rápida sucesión.

Referencia cruzada

La revolución del software se foto en el Capítulo 13. La ingeniería de software basada en componentes se presento en el Capítulo 27.

El determinismo de la información se refiere a la predecibilidad del orden y del tiempo de llegada de los datos. Un programa de análisis de ingeniería acepta datos que están en un orden predefinido, ejecuta el algoritmo(s) de análisis sin interrupción y produce los datos resultantes en un informe o formato gráfico. Se dice que tales aplicaciones son determinadas. Un sistema operativo multiusuario, por otra parte, acepta entradas que tienen un contenido variado y que se producen en instantes arbitrarios, ejecuta algoritmos que pueden ser interrumpidos por condiciones externas y produce una salida que depende de una función del entorno y del tiempo. Las aplicaciones con estas características se dice que son indeterminadas.

Algunas veces es difícil establecer categorías genéricas para las aplicaciones del software que sean significativas. Conforme aumenta la complejidad del software, es más difícil establecer compartimentos nítidamente separados. Las siguientes áreas del software indican la amplitud de las aplicaciones potenciales:

Software de sistemas. El software de sistemas es un conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistemas (por ejemplo: compiladores, editores y utilidades de gestión de archivos) procesan estructuras de información complejas pero determinadas. Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados. En cualquier caso, el área del software de sistemas se caracteriza por una fuerte interacción con el hardware de la computadora; una gran utilización por múltiples usuarios; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; unas estructuras de datos complejas y múltiples interfaces externas.

Software de tiempo real. El software que coordina/analiza/controla sucesos del mundo real conforme ocurren, se denomina de tiempo real. Entre los elementos del software de tiempo real se incluyen: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo, y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse la repuesta en tiempo real (típicamente en el rango de un milisegundo a un segundo).

Software de gestión. El proceso de la información comercial constituye la mayor de las áreas de aplicación del software. Los «sistemas» discretos (por ejemplo: nóminas, cuentas de haberes-débitos, inventarios, etc.) han evolucionado hacia el software de sistemas de información de gestión **(SIG)** que accede a una o más bases de datos que contienen información comercial. Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamientos de datos, las aplicaciones de software de gestión también realizan cálculo interactivo (por ejemplo: el procesamiento de transacciones en puntos de ventas).

Software de ingeniería y científíco. El software de ingeniería y científico está caracterizado por los algoritmos de «manejo de números». Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación automática. Sin embargo, las nuevas aplicaciones del área de ingeniería/ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (del inglés CAD), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a coger características del software de tiempo real e incluso del software de sistemas.

Software empotrado. Los productos inteligentes se han convertido en algo común en casi todos los mercados de consumo e industriales. El software empotrado reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. El software empotrado puede ejecutar funciones muy limitadas y curiosas (por ejemplo: el control de las teclas de un horno de microondas) o suministrar una función significativa y con capacidad de control (por ejemplo: funciones digitales en un automóvil, tales como control de la gasolina, indicadores en el salpicadero, sistemas de frenado, etc.).

Referencia Web

Se puede encontrar una de las mayores bibliotecas de *shareware/freeware* en **www.shareware.com**

Software de computadoras personales. El mercado del software de computadoras personales ha germinado en las pasadas dos décadas. El procesamiento de textos, las hojas de cálculo, los gráficos por computadora, multimedia, entretenimientos, gestión de bases de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas son algunas de los cientos de aplicaciones.

Software basado en Web. Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, CGI, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales). En esencia, la red viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquiera con un modem.

Software de inteligencia artificial. El software de inteligencia artificial (IA) hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en el conocimiento, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas, y los juegos son representativos de las aplicaciones de esta categoría.

1.3 SOFTWARE: ¿UNA CRISIS EN EL HORIZONTE?

Muchos observadores de la industria (incluyendo este autor) han caracterizado los problemas asociados con el desarrollo del software como una «crisis». Más de unos cuantos libros (por ejemplo: [GLA97], [FLO97], [YOU98a]) han recogido el impacto de algunos de los fallos mas importantes que ocurrieron durante la década pasada. No obstante, los mayores éxitos conseguidos por la industria del software han llevado a preguntarse si el término (crisis del software) es aún apropiado. Robert Glass, autor de varios libros sobre fallos del software, representa a aquellos que han sufrido un cambio de pensamiento. Expone [GLA98]: «Puedo ver en mis ensayos históricos de fallos y en mis informes de excepción, fallos importantes en medio de muchos éxitos, una copa que está [ahora] prácticamente llena.»

> **Cita:**
> La mayoría de los expertos están de acuerdo en que la manera más probable para que el mundo se destruya es por accidente. Ahí es donde nosotros entramos; somos profesionales de la informática, provocamos accidentes.
> **Nathaniel Borenstein**

La palabra crisis se define en el *diccionario Webster* como «un punto decisivo en el curso de algo, momento, etapa o evento decisivo o crucial». Sin embargo, en términos de calidad del software total y de velocidad con la cual son desarrollados los productos y los sistemas basados en computadoras, no ha habido ningún «punto crucial», ningún «momento decisivo», solamente un lento cambio evolutivo, puntualizado por cambios tecnológicos explosivos en las disciplinas relacionadas con el software.

Cualquiera que busque la palabra *crisis* en el diccionario encontrará otra definición: «el punto decisivo en el curso de una enfermedad, cuando se ve más claro si el paciente vivirá o morirá». Esta definición puede darnos una pista sobre la verdadera naturaleza de los problemas que han acosado el desarrollo del software.

Lo que realmente tenemos es una aflicción crónica[1]. La palabra *aflicción* se define como «algo que causa pena o desastre». Pero la clave de nuestro argumento es la definición del adjetivo *crónica:* «muy duradero *o* que reaparece con frecuencia continuando indefinidamente». Es bastante más preciso describir los problemas que hemos estado aguantando en el negocio del software como una aflicción crónica, en vez de como una crisis.

Sin tener en cuenta como lo llamemos, el conjunto de problemas encontrados en el desarrollo del software de computadoras no se limitan al software que «no funciona correctamente». Es más, el mal abarca los problemas asociados a cómo desarrollar software, cómo mantener el volumen cada vez mayor de software existente y cómo poder esperar mantenemos al corriente de la demanda creciente de software.

Vivimos con esta aflicción desde este día —de hecho, la industria prospera a pesar de ello—. Y así, las cosas podrán ser mejores si podemos encontrar y aplicar un remedio.

1.4 MITOS DEL SOFTWARE

Muchas de las causas de la crisis del software se pueden encontrar en una mitología que surge durante los primeros años del desarrollo del software. A diferencia de los mitos antiguos, que a menudo proporcionaban a los hombres lecciones dignas de tener en cuenta, los mitos del software propagaron información errónea y confusión. Los mitos del software tienen varios atributos que los hacen insidiosos: por ejemplo, aparecieron como declaraciones razonables de hechos (algunas veces conteniendo elementos verdaderos), tuvieron un sentido intuitivo y frecuentemente fueron promulgados por expertos que «estaban al día».

[1] Esta terminología fue sugerida por el profesor Daniel Tiechrow de la Universidad de Michigan en una conferencia impartida en Ginebra, Suiza, Abril, 1989.

Mitos de gestión. Los gestores con responsabilidad sobre el software, como los gestores en la mayoría de las disciplinas, están normalmente bajo la presión de cumplir los presupuestos, hacer que no se retrase el proyecto y mejorar la calidad. Igual que se agarra al vacío una persona que se ahoga, un gestor de software se agarra frecuentemente a un mito del software, aunque tal creencia sólo disminuya la presión temporalmente.

Mito. Tenemos ya un libro que está lleno de estándares y procedimientos para construir software, ¿no le proporciona ya a mi gente todo lo que necesita saber?

Realidad. Está muy bien que el libro exista, pero jse usa?.¿conocen los trabajadores su existencia?, jrefleja las prácticas modernas de desarrollo de software?, ¿es completo?, ¿está diseñado para mejorar el tiempo de entrega mientras mantiene un enfoque de calidad? En muchos casos, la respuesta a todas estas preguntas es «no».

Cita:
En ausencia de estándares significativos, una nueva industria como la del software pasa a depender de las costumbres.
Tom De Marco

Mito. Mi gente dispone de las herramientas de desarrollo de software más avanzadas, después de todo, les compramos las computadoras más modernas.

Realidad. Se necesita mucho más que el último modelo de computadora grande o de PC para hacer desarrollo de software de gran calidad. Las herramientas de ingeniería del software asistida por computadora (CASE) son más importantes que el hardware para conseguir buena calidad y productividad, aunque la mayoría de los desarrolladores del software todavía no las utilicen eficazmente.

Mito. Si fallamos en la planificación, podemos añadir más programadores y adelantar el tiempo perdido (el llamado algunas veces «concepto de la horda Mongoliana»).

Realidad. El desarrollo de software no es un proceso mecánico como la fabricación. En palabras de Brooks [BRO75]: «...añadir gente a un proyecto de software retrasado lo retrasa aún más». Al principio, esta declaración puede parecer un contrasentido. Sin embargo, cuando se añaden nuevas personas, la necesidad de aprender y comunicarse con el equipo puede y hace que se reduzca la cantidad de tiempo gastado en el desarrollo productivo. Puede añadirse gente, pero sólo de una manera planificada y bien coordinada.

Referencia Web
la red de gestión de proyectos de software en **www.spmn.com** puede ayudarle a desmitificar estos y otros mitos.

Mitos del Cliente. Un cliente que solicita una aplicación de software puede ser una persona del despacho de al lado, un grupo técnico de la sala de abajo, el departamento de ventas o una compañía exterior que solicita un software bajo contrato. En muchos casos, el cliente cree en los mitos que existen sobre el software, debido a que los gestores y desarrolladores del software hacen muy poco para corregir la mala información. Los mitos conducen a que el cliente se cree una falsa expectativa y, finalmente, quede insatisfecho con el que desarrolla el software.

Mito. Una declaración general de los objetivos es suficiente para comenzar a escribir los programas —podemos dar los detalles más adelante—.

Realidad. Una mala definición inicial es la principal causa del trabajo baldío en software. Es esencial una descripción formal y detallada del ámbito de la información, funciones, comportamiento, rendimiento, interfaces, ligaduras del diseño y criterios de validación. Estas características pueden determinarse sólo después de una exhaustiva comunicación entre el cliente y el analista.

Mito. Los requisitos del proyecto cambian continuamente, pero los cambios pueden acomodarse fácilmente, ya que el software es flexible.

Referencia cruzada
La gestión y control de cambio está tratada con detalle en el Capítulo 9

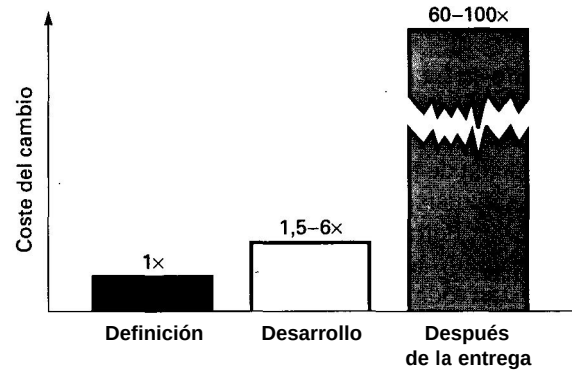


FIGURA 1.3. El impacto del cambio.

Realidad. Es verdad que los requisitos del software cambian, pero el impacto del cambio varía según el momento en que se introduzca. La Figura **1.3** ilustra el impacto de los cambios. Si se pone cuidado al dar la definición inicial, los cambios solicitados al principio pueden acomodarse fácilmente. El cliente puede revisar los requisitos y recomendar las modificaciones con relativamente poco impacto en el coste. Cuando los cambios se solicitan durante el diseño del software, el impacto en el coste crece rápidamente. Ya se han acordado los recursos a utilizar y se ha establecido un marco de trabajo del diseño. Los cambios pueden producir trastornos que requieran recursos adicionales e importantes modificaciones del diseño; es decir, coste adicional. Los

cambios en la función, rendimiento, interfaces u otras características, durante la implementación (codificación y prueba) pueden tener un impacto importante sobre el coste. Cuando se solicitan al final de un proyecto, los cambios pueden producir un orden de magnitud más caro que el mismo cambio pedido al principio.

Mitos de los desarrolladores. Los mitos en los que aún creen muchos desarrolladores se han ido fomentando durante 50 años de cultura informática. Durante los primeros días del desarrollo del software, la programación se veía como un arte. Las viejas formas y actitudes tardan en morir.

Mito. Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.

Realidad. Alguien dijo una vez: «cuanto más pronto se comience a escribir código, más se tardará en terminarlo». Los datos industriales [LIE80, JON91, PUT97] indican que entre el 60 y el 80 por ciento de todo el esfuerzo dedicado a un programa se realizará después de que se le haya entregado al cliente por primera vez.

Trabaja muy duro poro entender lo que tienes que hacer antes de empezar. No serías copoz de desarrollar codo detalle; por más que sepas, tomo el menor riesgo.

Mito. Hasta que no tengo el programa «ejecutándose», realmente no tengo forma de comprobar su calidad.

Realidad. Desde el principio del proyecto se puede aplicar uno de los mecanismos más efectivos para garantizar la calidad del software: *la revisión técnicaformal.* La revisión del software (descrito en el Capítulo 8) es un «filtro de calidad» que se ha comprobado que es más efectivo que la prueba, para encontrar ciertas clases de defectos en el software.

Mito. Lo único que se entrega al terminar el proyecto es el programa funcionando.

Realidad. Un programa que funciona es sólo una parte de una *configuración del software* que incluye muchos elementos. La documentación proporciona el fundamento para un buen desarrollo y, lo que es más importante, proporciona guías para la tarea de mantenimiento del software.

Muchos profesionales del software reconocen la falacia de los mitos descritos anteriormente. Lamentablemente, las actitudes y métodos habituales fomentan una pobre gestión y malas prácticas técnicas, incluso cuando la realidad dicta un método mejor. El reconocimiento de las realidades del software es el primer paso hacia la formulación de soluciones prácticas para su desarrollo.

RESUMEN

El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. En los pasados 50 años, el software ha pasado de ser una resolución de problemas especializada y una herramienta de análisis de información, a ser una industria por **sí** misma. Pero la temprana cultura e historia de la «programación» ha creado un conjunto de problemas que persisten todavía hoy. El software se ha convertido en un factor que limita la evolución de los sistemas informáticos. El software se compone de programas, datos y documentos. Cada uno de estos elementos componen una configuración que se crea como parte del proceso de la ingeniería del software. El intento de la ingeniería del software es proporcionar un marco de trabajo para construir software con mayor calidad.

Cuando te pones o pensar, no encuentras tiempo poro la disciplino de lo ingeniería del software, y te preguntas: «¿Tendré tiempo para poder hacerlo?»

REFERENCIAS

[BRO75] Brooks, F., *The Mytical Man-Month,* Addison-Wesley, 1975.

[DEJ98] De Jager, P., et al, *Countdown Y2K: Business Survival Planning for the Year 2000,* Wiley, 1998.

[DEM95] De Marco, T., *Why Does Software Cost So Much?,* Dorset House, 1995, p. 9.

[FEI83] Feigenbaum, E. A., y P. McCorduck, *The Fith Generation,* Addison-Wesley, 1983.

[FLO97] Flowers, S., *Software Failure, Management Failure-Amaicing Stories and Cautionary Tails,* Wiley, 1997 (?).

[GLA97] Glass, R. L., *Software Runaways,* Prentice Hall, 1997.

[GLA98] Glass, R. L.,«Is there Really a Software Crisis?», *IEEE Software,* vol. 15, n.º 1, Enero 1998, pp. 104-105.

[HAM93] Hammer, M., y J. Champy, *Reengineering the Corporation,* HarpperCollins Publisher, 1993.

[JON91] Jones, C., *Applied Software Measurement,* McGraw-Hill, 1991.

[KAR99] Karlson, E., y J. Kolber, *A Basic Introduction to Y2K:* How *the Year 2000 Computer Crisis Affects You?,* Next Era Publications, Inc., 1999.

[LEV95] Levy, S., «The Luddites Are Pack», *Newsweek*, 12 de Julio de 1995, p. 55.

[LEV99] Levy, S., «The New Digital Galaxy», *Newsweek*, 31 de Mayo de 1999, p.57.

[LIE80] Lientz, B., y E. Swanson, *Software Maintenance Management*, Addison Wesley, 1980.

[NAI82] Naisbitt, J., *Megatoends*, Warner Books, 1982.

[NOR98] Norman, D., *The Invisible Computer*, MIT Press, 1998.

[OSB79] Osborne, A., *Running Wild-The Next Industrial Revolution*, Osborne/McGraw-Hill, 1979.

[PUT97] Putnam, L., y W. Myers, *Industrial Strength Software*, IEEE Computer Society Press, 1997.

[STO89] Stoll, C., *The cuckoo's Egg*, Doubleday, 1989.

[TOF80] Toffler, A., *The Third Wave*, Morrow Publishers, 1980.

[TOF90] Toffler, A., *Powershift*, Bantam Publishers, 1990.

[YOU92] Yourdon, E., *The Decline and Fall of the American Programmer*, Yourdon Press, 1992.

[YOU96] Yourdon, E., *The Rise and Resurrection of the American Programmer*, Yourdon Press, 1996.

[YOU98a] Yourdon, E., *Death March Projects*, Prentice-Hall, 1998.

[YOU98b] Yourdon, E., y J. Yourdon, *Time Bomb* 2000, Prentice-Hall, 1998.

PROBLEMAS Y PUNTOS A CONSIDERAR

1.1. El software es la característica que diferencia a muchos productos y sistemas informáticos. Dé ejemplos de dos o tres productos y de, al menos, un sistema en el que el software, no el hardware, sea el elemento diferenciador.

1.2. En los años cincuenta y sesenta la programación de computadoras era un arte aprendido en un entorno básicamente experimental. ¿Cómo ha afectado esto a las prácticas de desarrollo del software hoy?

1.3. Muchos autores han tratado el impacto de la «era de la información». Dé varios ejemplos (positivos y negativos) que indiquen el impacto del software en nuestra sociedad. Repase algunas referencias de la Sección 1.1 previas a 1990 e indique dónde las predicciones del autor fueron correctas y dónde no lo fueron.

1.4. Seleccione una aplicación específica e indique: (a) la categoría de la aplicación de software (Sección 1.2.2) en la que encaje; (b) el contenido de los datos asociados con la aplicación; (c) la información determinada de la aplicación.

1.5. A medida que el software se difunde más, los riesgos para el público (debido a programas defectuosos) se convierten en una preocupación cada vez más significativa. Desarrolle un escenario realista del juicio final (distinto a Y2K) en donde el fallo de computadora podría hacer un gran daño (económico o humano).

1.6. Lea detenidamente el grupo de noticias de Internet **comp.risk** y prepare un resumen de riesgos para las personas con las que se hayan tratado Últimamente. Código alternativo: *Software Engineering Notes* publicado por la ACM.

1.7. Escriba un papel que resuma las ventajas recientes en una de las áreas de aplicaciones de software principales. Entre las selecciones potenciales se incluyen: aplicaciones avanzadas basadas en Web, realidad virtual, redes neuronales artificiales, interfaces humanas avanzadas y agentes inteligentes.

1.8. Los mitos destacados en la Sección 1.4 se están viniendo abajo lentamente a medida que pasan los años. Pero otros se están haciendo un lugar. Intente añadir un mito o dos mitos «nuevos» a cada categoría.

OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

Literalmente existen miles de libros escritos sobre software de computadora. La gran mayoría tratan los lenguajes de programación o aplicaciones de software, y sólo unos pocos tratan el software en sí. Pressman y Herron *(Software Sock*, Dorset House, 1991) presentaron una discusión (dirigida a no profesionales) acerca del software y del modo en que lo construyen los profesionales.

El libro, éxito de ventas, de Negroponte *(Being Digital*, Alfred A. Knopf, Inc., 1995) proporciona una visión de las computadoras y de su impacto global en el siglo XXI. Los libros de Norman [NOR98] y Bergman *(Information Appliances & Beyond*, Academic Pres/Morgan Kauffman, 2000) sugieren que el impacto extendido del PC declinará al mismo tiempo que las aplicaciones de información y la difusión de la programación conecten a todos en el mundo industrializado y casi todas las aplicaciones a la nueva infraestructura de Internet.

Minasi *(The Software Conspiracy: Why Software Companies Put Out Faulty Products, How They Can Hurt* ***You,*** *and What You Can Do*, McGraw-Hill, 2000) argumentó que el «azote moderno» de los errores del software puede eliminarse y sugiere formas para hacerlo. DeMarco (*Why Does Software Cost So Much?*, Dorset House, 1995) ha producido una colección de ensayos divertidos e interesantes sobre el software y el proceso a través del cual se desarrolla.

En Intemet están disponibles una gran variedad de fuentes de información relacionadas con temas de gestión y de software. Se puede encontrar una lista actualizada con referencias a sitios (páginas) web relevantes en **http://www.pressman5.com.**